Attorney Docket No. IBM/09B
Confirmation No. 1040

PATENT

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

Ex parte William Jon Schmidt

Appeal No. _____
Application No. 09/626,982

APPEAL BRIEF

OFFICIAL

PATENT

Attorney Docket No. IBM/09B
Confirmation No. 1040

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	William Jon Schmidt	Art Unit:	2122
Serial No.:	09/626,982	Examiner:	K. Gross
Filed:	June 27, 2000	Atty. Docket No.:	IBM/09B
For:	SKIP LIST DATA STORAGE DURING COMPILATION		

#9/E.O.T
Appeal
2-27-04

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

I. REAL PARTY IN INTEREST

This application is assigned to International Business Machines Corporation, of Armonk, New York.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

III. STATUS OF CLAIMS

Claims 12-19 and 24-35 are pending in the Application, with claim 12 being once amended. All pending claims stand rejected, and are now on appeal.

IV. STATUS OF AMENDMENTS

There have been no amendments filed subsequent to the final rejection (Paper 7).

V. SUMMARY OF INVENTION

The present invention relates to the initialization of a data flow analysis conducted as part of compiling source language into object code. Compilers which translate source language into

machine readable object code, frequently include an optimizer or optimizing module for enhancing the performance of the machine readable code.

Typically a program or procedure being compiled is broken down into a series of "statements", each of which contains zero or more "operands" or "data items". A data item may be "defined", meaning that it is given a value by the statement, or "used", meaning that its value is fed into the computation represented by the statement. For example, the statement "x = y + z" defines x and uses y and z. Optimization of a program often involves locating individual statements or groups of statements which can be eliminated or rewritten in such a way as to reduce the total number of statements in the program or in a particular flow path through the program. For example, a complicated expression might be computed at two distant points within the same procedure. If the variables used in the expression are not modified to contain different values between the first and second computations, the value can be computed only once, at the first point in the procedure, and saved in a temporary location for use at the second point in the procedure, thus avoiding recomputation at the second point. This particular form of optimization is known as "common (sub)expression elimination".

The main problem in optimizing a procedure is to determine at which points of the procedure various kinds of information are available. For example, to perform common (sub)expression elimination, it is necessary to know at which points the variables used by the procedure are modified. To determine such facts, a dataflow analysis is performed on the program.

To perform dataflow analysis, possible paths of execution through a procedure may be represented by a control flow graph (CFG). The CFG is a graph with one node for each basic block in the procedure. The CFG includes an arc from block A to block B if it is possible for block B to be executed immediately after block A has been executed. In such a case, B is called a "successor" of A, and A is called a "predecessor" of B.

After generating a CFG, optimization typically involves computing the properties of the statements in each block in the CFG. Often, a matrix of bits is used to identify and track these properties. Typically, the matrix is initialized by assuming that every statement is available in

every block in the CFG. Then, iterative passes are made through the CFG to correct the matrix, by altering bits to indicate statements that are not available in particular blocks.

In practice, most of the bits in the matrix are zero (i.e., the matrix is "sparse"). In a typical case where the bits in the matrix relate to the status of particular expressions, the matrix is typically sparse because, normally, specific expressions are only used or useful in a small portion of a procedure. A large, sparse bit matrix not only consumes large quantities of space, but also requires a large amount of time to repeatedly scan in the manner needed for complex dataflow analysis.

The present application describes an alternative representation for storing the status of properties, in the form of a skip list, which is claimed by the related U.S. Patent 6,117,185. This representation is substantially more efficient than a bit matrix, both in storage space required and in the access time required to find and alter a property for a given statement. However, this skip list representation can only efficiently store a sparsely populated set, and would be inefficient if the initialization of the dataflow analysis used the standard approach of assuming that every statement is available in every basic block.

Accordingly, the present divisional application is directed to a novel procedure for initializing entry and exit properties for basic blocks of a computer procedure, that does not assume, for example, that every statement is available in every basic block. In this novel method, the properties for the basic blocks are initialized systematically in a predetermined order. More particularly, the entry properties of a currently selected basic block, are copied from exit properties of a previously selected and processed basic block. Next, the exit properties for the currently selected basic block, are computed from the entry properties and the property modifications associated with the currently selected basic block. By passing through basic blocks using this novel method, the property sets for basic blocks can be initialized to an appropriate initial state, without resorting to the inefficient assumption that every statement is available in every basic block.

Dependent claims recite various specifics of the claimed concepts, such as that the entry and exit properties are sets of expressions available upon entry and exit from the basic block, and that the property modifications are expressions generated and killed by the basic block. Also,

further dependent claims recite the additional feature that expressions not found in the sets of expressions available upon exit from all previously selected and processed control flow predecessors of the currently selected basic block, are removed from the set of expressions available upon entry to the currently selected basic block.

VI. ISSUES

A. Whether all of claims 12-19 and 24-35 are anticipated by or rendered obvious by Aho, "Compilers: Principles, Techniques and Tools".

B. Whether claims 24-31 define statutory subject matter under 35 U.S.C. 101.

VII. GROUPING OF CLAIMS

Claims 19 and 31 stand or fall separately from the other claims for the reason that claims 19 and 31 recite the removal of properties based upon the unavailability of those properties in any control flow predecessor basic blocks, which is an independently patentable concept from the concepts of the remaining claims.

VIII. ARGUMENT

The Examiner's Office Action rejects all claims as anticipated by or obvious in light of the Aho book, and specifically Fig. 10.21 of that book. The Examiner relies upon Aho's description of relationships between in[S], out[S], gen[S] and kill[S], and an analogy to the steps of claim 12, 24 and 31.

Applicant notes, however, that claims 12, 24 and 31, and therefore all claims, are directed to a "method of calculating approximations of sets of entry and exit properties ... prior to performance of an iterative dataflow analysis" – and the iterative dataflow analysis is described as one which "does not increase the membership of [the] sets".

Aho provides algorithms for computing in[S] and out[S], in section 10.6, and these are distinctly different from what is claimed. The Aho book provides several examples of dataflow

analysis, but none match what is claimed. Generally, there are two categories of dataflow algorithms described:

- An algorithm that initializes in[S] and out[S] to have no members, and then performs an iterative process which adds members to in[S] and out[S]. This is the case in Algorithm 10.2 on page 625 which "start[s] with the 'estimate' in[B] = ⊘ for all B and converg[es] to the desired values of in and out".

- An algorithm that initializes in[S] and out[S] to have all possible members, and then performs an iterative process which deletes members from in[S] and out[S]. This is the case in Algorithm 10.3 on page 631 which starts with an "initial estimate [which] is too large", computed by "out[B] = U - e_kill[B]"; that is, Algorithm 10.3 starts by assuming that every expression except those killed by B are in out[B], and then eliminates expressions from this large universe, through iterations of the process.

Applicant submits that the claimed invention is an algorithm for initialization prior to a dataflow analysis, which is neither of the above. Unlike the first of Aho's proposals, the claimed invention uses an iterative dataflow analysis that does not increase the membership of sets. And unlike the second of Aho's proposals, the claimed invention does not assume an excessively large initial membership of the initial values of in[S] and out[S] (which, in algorithm 10.3, can be as large as the entire universe U of expressions).

As explained in the Application at page 20, a gross overestimation of in[S] and out[S], as is typically used in dataflow analysis algorithms like algorithm 10.3, does not present a problem with a conventional bit-vector representation for in[S] and out[S]. However, when there is an unconventional representations of in[S] and out[S], such as the skip-list representation invented by the inventor and disclosed in this application (and claimed by the parent application, now U.S. Patent 6,117,185), and this unconventional representation is "not efficent in representing densely populated sets", a gross overestimation of in[S] and out[S] can be debilitating. In the presently claimed invention, therefore, "a novel methodology is used to initialize the sets in[.] and out[.]. This methodology is based on the recognition that (1) the dataflow analysis conducted in accordance with Fig. 4 only reduces the number of members in these sets, and (2) it is only necessary that in[.] and out[.] initially include all members that might be included after dataflow

analysis. If a simplified analysis of the blocks B in the CFG can identify all of the members that might possibly be included in the in[.] and out[.] sets after a complete data flow analysis, then only those members that might possibly be included need be included when the in[.] and out[.] sets are initialized."

The present invention provides just such an initialization process that allows the use of the unconventional and advantageous skip-list representation that is the subject of the sibling patent 6,117,185. Applicant submits that nothing in the Aho confronts or would provide any initialization process of this kind, as Aho assumes a bit-vector representation of the sets in[S] and out[S] (see page 618, under "Representation of Sets"). Specifically, Aho does not provide an initialization process that is at all analogous to the steps recited in claim 12.

Applicant notes that the preamble of each of claims 12, 24 and 31 recites "calculating approximations of ... properties ... prior to performance of an iterative dataflow analysis"; thus, the claims are directed specifically to "calculating approximations" "prior to performance of an iterative dataflow analysis". That is, the claims are directed to what happens before dataflow analysis. The comparable steps in the Aho prior art, are the initial steps of assuming that in[.] and out[.] include all members, or no members at all – which is completely unlike the steps recited in claims 12, 24 and 31: "copying" properties from one basic block to another, and "modifying" properties in accordance with property modifications in a basic block.

To ensure the claims can only be read to describe what is done "prior to performance of an iterative dataflow analysis", Applicant included in each of the independent claims, a separate step of "performing iterations of [the] iterative dataflow analysis". Since the step of "performing iterations of [the] iterative dataflow analysis" is positively recited in each claim, the "copying" and "modifying" steps of each claim must be done in addition to "iterative dataflow analysis". Thus, the only activity in Aho that can be analogized to the "copying" and "modifying" steps, are the two irrelevant approaches of assuming in[.] and out[.] include all members, or none at all.

Notwithstanding this claim language structure, and the logical consequences that flow from it, the Examiner's final rejection asserts anticipation by Aho, based upon an analogy between steps taken during the Aho iterative dataflow analysis, and the "copying" and "modifying" steps in the claimed initialization process that occurs in addition to "iterative

dataflow analysis". The Examiner's explanation for this is that "there is no language ... that indicates ordering of the steps ... There is no way to tell which steps take place 'prior to performance of an iterative dataflow analysis' and which take place during the analysis itself. Without proper claim language indicating order, it can be assumed that any steps can be performed at any time."

As is apparent from the above discussion, however, the "copying" and "modifying" steps of the claims cannot be interpreted to occur "during the analysis itself", for the reason that the "analysis itself" is its own step, i.e., the separate step of "performing iterations of said iterative dataflow analysis". The claim thus requires steps of "copying" and "modifying" that occur in addition to the "analysis itself", and this Aho clearly does not show. In Aho, the only actions outside of the "analysis itself" is the assumption that in[.] and out[.] include all members, or none at all.

The Examiners' requirement for language indicative of order, is also not well founded. Firstly, the preambles of the independent claims clearly state that the claim steps are for calculating approximations "prior to performing an iterative dataflow analysis", thus establishing the order. Furthermore, even if the claims did not recite the order of their steps, the claim language makes clear that the steps of "copying" and "modifying" occur in addition to the "iterative dataflow analysis", and so could not be part of the "iterative dataflow analysis".

Applicant respectfully submits the Examiner's rejection on prior art is erroneous, and requests reversal.

Separately from the above reasons for reversal, claims 19 and 31 have independent bases for allowance. Claims 19 and 31 recite the removal of properties from a basic block, based upon the unavailability of those properties in any control flow predecessor basic blocks. This is an independently patentable concept from the concepts of the remaining claims, and nothing analogous has been identified in any part of Aho. The Examiner's rejection of claim 19 (recited in the Office Action of February 3, 2003) is that Aho "teaches dead code elimination" and that "according to dead code elimination, [an unused] expression should be removed from the entry properties of the currently selected basic block". With all due respect, this is a nonsequitur. Dead code elimination involves removing statements from a block that are not used. It has

nothing to do with removing entry and exit properties from dataflow blocks. Furthermore, dead code elimination relates to analyzing code in one block to find code that is never used; it has nothing to do with identifying properties that are unavailable in potentially many control flow predecessor blocks, to identify unavailable properties. The two are simply unrelated. Applicant respectfully submits that the rejection of claims 19 and 31 is unsound and requests reversal.

The Examiner's rejection for nonstatutory subject matter, is not well understood. The substance of the rejection is that "claim 24 teaches a computer system with compiler and optimizer to perform the method of calculating approximations of sets of entry and exit properties for each basic block of a computer procedure, however the compiler and optimizer of the computer system are not described as residing on any computer hardware medium."

Claim 24 that is at issue recites: "A computer system operating a compiler". Applicant is at a loss to understand how "a computer system" is not statutory subject matter. Contrary to the Examiner's assertion, the claim clearly recites an apparatus ("hardware medium") – the apparatus is a "computer system". This is clearly statutory subject matter. Applicant submits the Examiner's rejection under 35 U.S.C. 101 is erroneous, and requests reversal.

IX. CONCLUSION

In conclusion, Applicants respectfully request that the Board reverse the Examiner's rejections of claims 12-19 and 24-35, and that the Application be passed to issue. If there are any questions regarding the foregoing, please contact the undersigned at 513/241-2324. Moreover, if any other charges or credits are necessary to complete this communication, please apply them to Deposit Account 23-3000.

Respectfully submitted,

WOOD, HERRON & EVANS, L.L.P.

Date: Feb. 24, 2004

By: _____
Thomas W. Humphrey
Reg. No. 34,353

2700 Carew Tower
441 Vine Street
Cincinnati, Ohio 45202
(513) 241-2324

APPENDIX A: CLAIMS ON APPEAL (S/N 09/626,982)

12. (Previously Amended) A method of calculating approximations of sets of entry and exit properties for each basic block of a computer procedure prior to performance of an iterative dataflow analysis, each basic block being associated with a set of entry and exit properties, as well as property modifications caused by the basic block, wherein each iteration of the iterative dataflow analysis process does not increase the membership of said sets, the method comprising selecting and processing each basic block in a predetermined order by

copying into the set of entry properties of a currently selected basic block, the exit set of properties from a previously selected and processed basic block,

modifying the set of entry properties of the currently selected basic block in accordance with the property modifications caused by the currently selected basic block, to generate the exit properties for the currently selected basic block, and

performing iterations of said iterative dataflow analysis.

13. (Original) The method of claim 12, wherein said predetermined order is a depth-first order.

14. (Original) The method of claim 12, wherein said entry and exit properties comprise expressions available upon entry and exit from each basic block.

15. (Original) The method of claim 14, wherein said modifications caused by a basic block comprise expressions generated by the basic block.

16. (Original) The method of claim 15, wherein said modifications caused by a basic block further comprise expressions killed by the basic block.

17. (Original) The method of claim 14, wherein said modifications caused by a basic block comprise expressions killed by the basic block.

18. (Original) The method of claim 12, wherein the entry properties of the currently selected basic block are copied from the exit properties of a control flow predecessor of the currently selected basic block.

19. (Original) The method of claim 12, wherein processing the current basic block further comprises removing from the entry properties of the currently selected basic block, any properties not found in the exit properties of all previously selected and processed basic blocks which are control flow predecessors of the currently selected basic block.

24. (Previously Added) A computer system operating a compiler including an optimizer for calculating approximations of sets of entry and exit properties for each basic block of a computer procedure prior to performance of an iterative dataflow analysis, each basic block being associated with a set of entry and exit properties, as well as property modifications caused by the basic block, wherein each iteration of the iterative dataflow analysis process does not increase the membership of said sets, the optimizer selecting and processing each basic block in a predetermined order by

copying into the set of entry properties of a currently selected basic block, the exit set of properties from a previously selected and processed basic block,

modifying the set of entry properties of the currently selected basic block in accordance with the property modifications caused by the currently selected basic block, to generate the exit properties for the currently selected basic block, and

performing iterations of said iterative dataflow analysis.

25. (Previously Added) The computer system of claim 24, wherein said predetermined order is a depth-first order.

26. (Previously Added) The computer system of claim 24, wherein said entry and exit properties comprise expressions available upon entry and exit from each basic block.

27. (Previously Added) The computer system of claim 26, wherein said modifications caused by a basic block comprise expressions generated by the basic block.

28. (Previously Added) The computer system of claim 27, wherein said modifications caused by a basic block further comprise expressions killed by the basic block.

29. (Previously Added) The computer system of claim 26, wherein said modifications caused by a basic block comprise expressions killed by the basic block.

30. (Previously Added) The computer system of claim 24, wherein the entry properties of the currently selected basic block are copied from the exit properties of a control flow predecessor of the currently selected basic block.

31. (Previously Added) The computer system of claim 24, wherein processing the current basic block further comprises removing from the entry properties of the currently selected basic block, any properties not found in the exit properties of all previously selected and processed basic blocks which are control flow predecessors of the currently selected basic block.

32. (Previously Added) A program product comprising:

(a) a program defining the implementation of an optimizer for a compiler which calculates approximations of sets of entry and exit properties for each basic block of a computer procedure prior to performance of an iterative dataflow analysis, each basic block being associated with a set of entry and exit properties, as well as property modifications caused by the basic block, wherein each iteration of the iterative dataflow analysis process does not increase the membership of said sets, the program product comprising instructions for selecting and processing each basic block in a predetermined order by

copying into the set of entry properties of a currently selected basic block, the exit set of properties from a previously selected and processed basic block,

modifying the set of entry properties of the currently selected basic block in accordance with the property modifications caused by the currently selected basic block, to generate the exit properties for the currently selected basic block, and

performing iterations of said iterative dataflow analysis; and

(b) a signal bearing media bearing the program.

33. (Previously Added) The program product of claim 32, wherein said predetermined order is a depth-first order.

34. (Previously Added) The program product of claim 32, wherein the signal bearing media is a transmission type media.

35. (Previously Added) The program product of claim 32, wherein the signal bearing media is a recordable media.